Bluetooth Now or Low Energy: Should BLE Mesh Become a Flooding or Connection Oriented Network?

Yuri Murillo[1], Brecht Reynders[1], Alessandro Chiumento[1],
Salman Malik[2], Pieter Crombez[2] and Sofie Pollin[1]
[1] KU Leuven, Department of Electrical Engineering, Kasteelpark Arenberg 10, 3001 Heverlee, Belgium
[2] Televic Healthcare, Leo Bekaertlaan 1, 8870 Izegem, Belgium.
Email: {yuri.murillo, brecht.reynders}@esat.kuleuven.be

***Abstract*—Bluetooth Low Energy (BLE) represents the low-power, low-cost extension of the Bluetooth communication technology envisioned for the Internet of Things. Mesh protocols on top of BLE are currently emerging and the standard is currently being released. This paper first proposes a detailed measurement based comparison of two mesh approaches that fit within BLE operation: flooding and connection oriented networking. Using metrics such as packet delivery ratio (PDR), end-to-end delay and power consumption we conclude that the optimal mesh approach depends on the application. It is shown that for a comparable performance in terms of PDR and overhead, flooding can trade a lower end-to-end delay for a higher power consumption when compared to the connected mesh. We then propose an architecture, called Bluetooth Now, that is able to automatically switch the network between the two based on message priority. Our measurement results confirm the reliable delivery of important and urgent data sent using the Bluetooth Now paradigm, while saving battery life when transmitting non-time critical messages.**

I. INTRODUCTION

Bluetooth Low Energy (BLE) represents the low-power extension of Bluetooth envisioned for the Internet of Things (IoT). BLE was originally introduced in the Bluetooth Core Specification 4.0 and has received interest from both industrial and academic communities. The coverage of BLE for an indoor scenario has been studied in [1]; the results show that BLE devices have limited range and a large number of nodes are needed to cover office environments, so multi-hop protocols would overcome such range issues. The first version of BLE mesh has recently been released [2], although no commercially available node supports it yet nor any use case has been reported at the time of writing this paper.

BLE allows nodes to communicate either by advertising or connection packets. In the former a device simply broadcasts and any neighbor in range receives the data. In the latter two BLE nodes establish a connection after a handshake and then the packet is transmitted only between themselves [2]. The vast majority of proprietary BLE mesh protocols, as well as the official standard, make use of advertising messages. This connectionless scheme is able to distribute packets quickly over the network but has limited payload and the transmission reliability cannot be guaranteed [3]. Data is propagated through the network with a flooding mechanism, ensuring widespread delivery but at great overhead cost [4]. The implementation of connected BLE mesh networks in which the nodes route the packet to their neighbours has witnessed slow adoption from the industrial community. As far as the authors are aware, there exists only one available open source protocol, called FruityMesh [5].

As a primary contribution, this paper provides a fair comparison between connected mesh routing and broadcast flooding for multi-hop BLE by means of a thorough measurement campaign on single source, static, small scale networks. The metrics considered are the packet delivery ratio (PDR), end-to-end latency and power consumption. This comparison may then help to fine tune the current BLE mesh standard, as the trade off between energy consumption, delay, reliability and dimensionality of the network are the main challenges for IoT networks. To the best of the authors' knowledge, no previous work on measurement based comparison of BLE advertising versus connected performance exists.

As a secondary contribution, the implementation of Bluetooth Now is proposed. This architecture uses the connected low power mode by default but switches to the flooding one to ensure low latency of urgent messages.

The remainder of this work is structured as follows. Section II presents an overview of BLE mesh literature. Section III introduces the chosen flooding and routing schemes. Section IV depicts the measurement set-up and discusses the network metrics used for the comparison. In Section V the measurement results are presented. After this, Section VI presents the Bluetooth Now architecture. Finally, Section VII concludes this work.

II. RELATED WORK

Connected BLE routing: A connection oriented BLE mesh network is introduced in [6], where the authors present a Direct Acyclic Graph (DAG) based routing scheme to build the network. This scheme requires a central controller and continuous exchange of data from each node in order to maintain the graph up to date.

In [7] a BLE mesh protocol based on 6LowPAN is presented. 6LowPAN allows low-power nodes to use the

powerful and well known IP networking protocol, at the cost of large overhead compared to regular BLE.

Flooding based BLE routing: In [8] the authors present BLEmesh, a protocol based on broadcasting which uses advertising packets. This scheme relies on precomputed broadcasting attempts based on the probability that most nodes receive such transmissions. The results offered are only based on simulations.

In [9] the well known CSRmesh broadcasting mesh protocol is evaluated using both measurements and simulations. The results show that broadcasting mesh networks can achieve high packet delivery ratio as long as the throughput is maintained low.

III. BLE Mesh Networking

This section introduces the flooding and routing mechanisms considered: the Trickle algorithm and FruityMesh, respectively.

A. Flooding mesh: the Trickle algorithm

In this work Trickle [10] is chosen for the flooding mesh scheme as it is already a well-known and established network protocol. The justification is that the official BLE mesh protocol is based on a flooding scheme and is similar to Trickle in its operation, but was not yet released at the time of writing this paper.

In Trickle two kind of parameters are defined: design-time and run-time parameters. The design-time parameters are the limits of the Trickle Interval I_{MIN} and I_{MAX} and the redundancy constant k, which represents the threshold to decide whether to re-broadcast a message or not. The run-time parameters are the Trickle Interval I, the redundancy counter c and the transmission time t. During I, the node has a listen-only period from $[0, I/2)$ and a transmission period from $[I/2, I]$ in which t is chosen randomly from $t \in U(I/2, I)$. At any point in time, c is incremented every time the latest message is overheard, and if $c > k$ at time t then such message is not re-broadcasted by the node. When the interval I expires, it is doubled until it reaches I_{MAX} and c is reset to 0. If a new message different from the previous one is detected, the Trickle Interval is reset to I_{MIN} and the whole procedure is repeated.

B. Routing mesh: FruityMesh

The connection based mesh scheme chosen is FruityMesh [5], as it is the main open source implementation available. The algorithm is based on neighbour-only routing, where no routing tables are stored but a connection is established between two nodes and kept open until the packets reach their final destination. The network is created through a clustering process and the path from source to sink is randomly formed.

Detailed explanation of FruityMesh is outside the scope of this work and the interested reader is invited to read more and try the protocol [5].

IV. Measurement Setup and Performance Metrics

This section describes the setup used for the measurement campaign of the two BLE mesh flavors, as well as the values of their main parameters and the metrics used for measuring network performance.

A. Scenario

In order to compare Trickle with FruityMesh a multi-hop scenario is chosen, where all measurements are taken indoor in a regular office environment. The topology of the network is depicted in Figure 1.

In such scenario nine nodes are used: two nodes act as sink or source while the remaining seven are placed in mesh configuration. The only device generating traffic is the source and the packets are retransmitted by the nodes until they reach the sink. In order to ensure that both methods suffer from a similar environment in terms of interference and network activity, the different measurement runs are done in an alternate fashion: a Trickle run is followed by a FruityMesh run and vice-versa. For each run a total of 256 packets are generated by the source with rates of 1, 5 and 10 packets per second. Each data rate is repeated for 20 runs in order to average the results, leading to a total of 5120 packets transmitted per source rate.

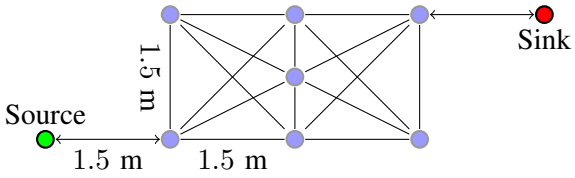

Fig. 1: Measurement setup. All vertical and horizontal distances are set to 1.5 m, except for the central node. The only forced connections (either Tricke or FruityMesh) are the ones of the source and sink, but within the mesh network all seven nodes are able to communicate with each other.

B. Hardware and Implementation

The nRF52 development boards manufactured by Nordic Semiconductor have been used in this work[11].

The boards are programmed with a pre-compiled binary file that contains the whole BLE protocol stack, known as the Softdevice. The two mesh protocols then run as applications on top of such stack. This approach decreases complexity in the development and implementation of the mesh protocols but gives reduced freedom when trying to set BLE specific parameters given the closed nature of the Softdevice.

A heavily modified version of the FruityMesh open code is implemented for this work. It supports not only regular FruityMesh but also Trickle. Commands can be introduced through a terminal session to the board in order to generate packets, synchronize the clocks of all

nodes, show statistics of the network or switch between Trickle or FruityMesh modes. When one node switches between mesh modes it broadcasts this change to the rest of the network, so all nodes overhear such packet and change their mesh modes accordingly. Further explanation of this architecture is given in Section VI.

C. Mesh Parameters and Packet Format

The transmission power for all nodes is set to $P_{tx} = -20$ dBm and their buffer sizes to 15 packets.

In Trickle the redundancy constant $k = 5$ in order to ensure a noticeable but moderate number of packets over the network. The values of $I_{MIN} = 160$ ms and $I_{MAX} = 22$ minutes are set to allow a fast propagation.

In FruityMesh each node has up to one master and three slave connections. The BLE connection interval is set to 100 ms, which results in a slightly different interval length than Trickle but is the closest one allowed by the Softdevice. Additionally, four retransmission attempts are allowed before a packet is discarded.

Each mesh packet has a size of 20 bytes (as shown in Figure 2) and is encapsulated into a BLE packet. The header includes a TYPE flag for identification and fragmentation purposes as well as the source and destination addresses. The payload consists of a TAG to distinguish the kind of packet (control or Trickle/FruityMesh data), a PID sequence number, a HOP counter, a timestamp for computing the delay and synchronizing the clocks of all nodes and finally a data field.

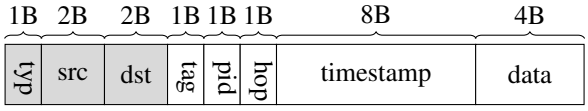

Fig. 2: Mesh packet with header (gray) and payload (white).

D. Metrics

1) General metrics: The metrics to compare the two BLE schemes are listed below, and all of them are computed at the sink node.

Packet Delivery Ratio: The PDR is the ratio of unique received packets at the sink $N_{\text{Rx,Sink}}$ over the number of packets transmitted by the source $N_{\text{Tx,Source}}$,

$$\text{PDR} = \frac{N_{\text{Rx,Sink}}}{N_{\text{Tx,Source}}}. \tag{1}$$

End-to-end delay: Once all nodes have their clocks synchronized (with accuracy in the order of a millisecond), this metric is computed as the time difference between the instant when the packet is generated at the source $t_{\text{Gen,Source}}$ and correctly read by the sink $t_{\text{Rx,Sink}}$,

$$\text{D}_{\text{end}} = t_{\text{Rx,Sink}} - t_{\text{Gen,Source}}. \tag{2}$$

Power Consumption: Based on the nRF52 datasheet [11] and the nRF52 online power profiler [12] the power consumption of a mesh interval per node can be estimated, assuming that one packet is received and transmitted per interval. In both mesh schemes the different events occurring within an interval multiplied by their duration will result in the average charge consumed, which is normalized by the total duration of the interval.

Three terms contribute to the average charge consumed in a FruityMesh event. The first one, namely I_{proc} accounts for all the processing involved (radio and CPU wake-up, packet processing, etc), while the second one describes the current consumption of receiving and transmitting a packet, defined as $I_{\text{radio}} = I_{\text{Rx}} + I_{\text{switch}} + I_{\text{Tx}}$. Additionally, a node exchanges keep-alive messages with its M connected neighbors, each one contributing with I_{conn}. Therefore, the average FruityMesh power consumption per interval T_{FM} can be expressed as

$$\text{P}_{\text{FM}} = V \frac{(I_{\text{proc}} + I_{\text{radio}} + M I_{\text{conn}})\, T_{\text{events}}}{T_{\text{FM}}}. \tag{3}$$

Trickle listens for the whole interval and the packet is broadcasted in all three advertising channels, so the power consumption per interval T_{TR} can be expressed as

$$\text{P}_{\text{TR}} = V \frac{(I_{\text{proc}} + 3I_{\text{radio}})\, T_{\text{events}} + I_{\text{Rx}} T_{\text{TR}}}{T_{\text{TR}}}. \tag{4}$$

2) FruityMesh specific metrics:

Number of Hops: Since the FruityMesh network is automatically generated, the path that the packet follows will be different every time the network is built. Therefore, the number of hops H will also be different.

Convergence delay: The convergence delay D_{conv} is defined as the time between turning on all nodes and having a fully-connected, single cluster network.

V. Measurement Results

This Section depicts the results obtained from the measurement campaign depicted in Figure 1.

1) General metrics:

Packet Delivery Ratio: Figure 3 presents the packet delivery ratio for the multi-hop network when 1, 5 or 10 packets per second are generated by the source.

The first noticeable result is that the FruityMesh PDR is not always 1 due to packet drops, which can be caused by two reasons. The first one may be excessive retransmission attempts without correctly receiving the packet in any of the intermediate links between source and sink due to channel impairments. The second one may be packet drop due to buffer overflow. Observe that both effects are strongly related to each other: the former will accentuate the later, as more and more packets will be queued up in the buffer while trying to correctly transmit the current one lost due to channel conditions. As the source maintains a constant packet generation rate, this will increase the probability of buffer overflow,

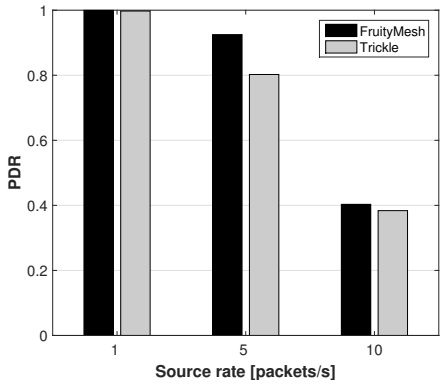

Fig. 3: Packet Delivery Ratio. Both mesh schemes show a degrading performance with source rates due to packet drops.

which will increase the probability of again needing a retransmission to deliver the packet. Therefore, this vicious circle will be specially noticeable when the number of hops H is high and/or the packet generation rate is high.

Additionally, the Trickle PDR is always lower than the one of FruityMesh due to the lack of retransmissions.

End-to-end delay: The average end-to-end delay D_{end} is presented in Figure 4. As the source rate increases the FuityMesh delay shows an extremely high increase when compared to the Trickle one. which remains constant.

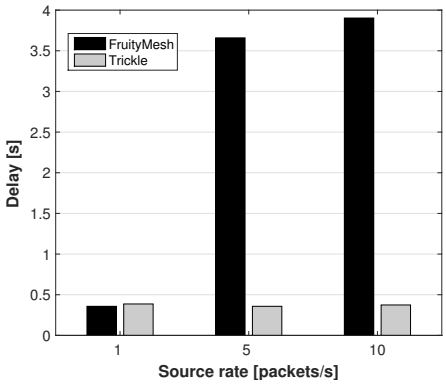

Fig. 4: End to end delay. FruityMesh suffers from increased delay compared to Trickle with high source rate due to the packet traveling through all buffers of the intermediate nodes.

For the case of FruityMesh the buffers become completely full when the source rate increases, and thus the average delay of the packet becomes at least the delay of advancing through all positions of the buffer multiplied by the number of hops in the path. Note that for a connection interval of 100 ms and a buffer size of 15 packets this means a delay of 1.5 s per node.

In Trickle the packets reach every node in the network: they travel through every path between source and sink and thus also the shortest possible path, ensuring the lowest end-to-end delay.

Power Consumption: For FruityMesh the average number of connections per node is $M = 2$, $V = 3$ V and $T_{\text{FM}} = 100$ ms. Based on the nRF52 power profiler, $I_{\text{proc}} = 11.4$ mA, $I_{\text{radio}} = 16.4$ mA, $I_{\text{conn}} = 54.9$ mA and $T_{\text{events}} = 2.4$ ms. Therefore the average power consumption is $\text{P}_{\text{FM}} = 9.4$ mW.

For Trickle $I_{\text{proc}} = 6.5$ mA, $I_{\text{radio}} = 45.3$ mA, $T_{\text{events}} = 4.4$ ms, $I_{\text{Rx}} = 5.6$ ms and finally $T_{\text{TR}} = I_{MIN} = 160$ ms. This results in $\text{P}_{\text{TR}} = 28.5$ mW.

Note that the power consumption of both mesh schemes is extremely high compared to regular BLE and exposes the challenge of designing a new mesh paradigm compatible with current power consumption needs.

2) FruityMesh specific metrics:

Number of hops: Table I summarizes the measured hop count behavior. Note that a hop count of 2 explains the average delay of Figure 4.

TABLE I: Number of hops H statistics

Max	Min	Mode
5	2	2

Convergence delay: The convergence delay boxplot for all measurement runs is presented in Figure 5.

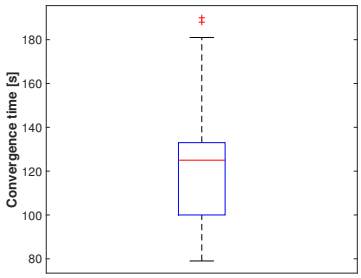

Fig. 5: Convergence delay D_{conv}, showing high variance between runs. This is unacceptable for low latency applications.

It can be seen that the network requires a considerable amount of time to converge to a stable configuration. The average network convergence delay is $\overline{D}_{\text{conv}} = 126$ s. For low-latency applications where the packets need to be delivered to the sink as fast as possible, FruityMesh becomes a weak choice as not only the convergence delay is high but also the end-to-end delay can be large.

VI. BLUETOOTH NOW

In light of the results from the measurement campaign, there is no mesh approach for BLE that can offer optimal performance for all metrics simultaneously. On the one hand, when the source of data corresponds to a sporadic and urgent application a flooding mechanism should be chosen in order to ensure minimal end-to-end delay. On the other hand, periodic and non-time sensitive data should propagate through a connection-oriented mesh in order to reduce its energy consumption. Based on

the FruityMesh architecture this Section proposes the Bluetooth Now paradigm, where both schemes coexist.

A. Application

A real life application where the two sources of data previously introduced coexist is a hospital environment.

In this scenario patients have their body values constantly monitored, like heart-rate, blood pressure and body temperature. This represents the periodical and non-time sensitive information for the network.

In case of an emergency an alarm may be sent over the mesh network to inform the nurses about the situation. This represents the sporadic and urgent type of data.

B. The Bluetooth Now Paradigm

Motivated by this scenario the Bluetooth Now paradigm is proposed, where the network operates in connection mode while forwarding the monitoring data but switches to flooding for blasting the alarm through the mesh in order to ensure minimal end-to-end delay. After this flood, the network goes back to connected mode to deliver the steady state information once again. The following discussion introduces the protocol.

Similarly to FruityMesh, the Bluetooth Now architecture consists of several modules that provide different levels of functionality, depicted in Figure 6.

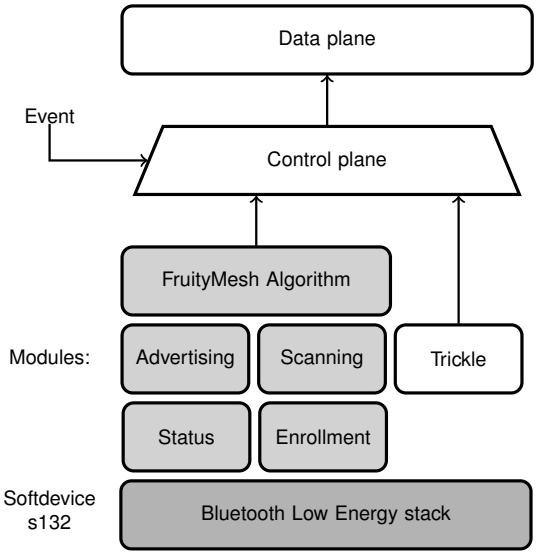

Fig. 6: Proposed architecture. Gray blocks correspond to the modified FruityMesh architecture, while white blocks correspond to blocks specifically implemented for BLE Now.

Modules: The reader interested in an introduction to the already existing FruityMesh modules is referred to the wiki [5]. Trickle is implemented as an additional module inside the FruityMesh project and it is always enabled irrespective of the mesh mode selected by the control plane. The reason behind this design choice is that the order for switching between one mesh mode or the other is done through a Trickle message. Note however that at all times the data packets are sent only using one of the two mesh modes.

Control plane: The control plane is responsible of selecting between FruityMesh or Trickle. It is event-triggered, either manually sending commands through a terminal or by overhearing the switching order from a neighbor. When selecting the mesh mode, the control plane does the switching and communicates this action to the Softdevice (so the correct BLE mechanism is used) and the data plane (so the switching order can be broadcasted). Additionally, the control plane can offer advanced functionality, such as synchronizing all nodes in the network by means of broadcasting timestamped packets and adjusting their clocks accordingly.

C. Performance Measurements

In order to showcase the improved performance that Bluetooth Now offers in the considered scenario a similar multi-hop network to the one of Figure 1 is set up. In such, an alarm node is positioned next to the source. The source is set to generate traffic at a rate of 5 packets/s in order to emulate the amount of traffic delivered by all the sensor nodes through the network. Once the network is in steady state, the alarm message is also generated at a rate of 5 packets/s. This packet needs to be delivered to the sink as soon as possible, but depending on the mesh mode used the delay may be drastically different.

When the network is in static Trickle mode, the alarm simply broadcasts its data and two parallel Trickle floods run within the network. When the network is in static FruityMesh mode, the alarm needs to join the network, find a route to the sink and send the packet through that specific route. When using Bluetooth Now, the network is in FruityMesh mode but the alarm first sends the switching order and then broadcasts the data using Trickle. After doing so, it sends the switching order again and the network resumes FruityMesh operation.

Note that two different types of delay contribute to the overall alarm delay $D_{alarm} = D_{init} + D_{end}$. First, the initialization delay D_{init} accounts for the alarm node entering the network: in the case of Trickle $D_{init} = 0$ as the alarm only needs to start broadcasting; in the case of FruityMesh D_{init} represents the enrollment time and finding the route to the sink; in the case of Bluetooth Now D_{init} accounts for the network switching from FruityMesh mode to Trickle mode. Second, once the alarm has joined the network the end-to-end delay D_{end} defined in Section IV measures the amount of time that the packet needs for traveling through the network.

The alarm delay is averaged over 10 consecutive realizations of each of the three considered mesh technologies, and the results are depicted in Figure 7. As can be seen, from the readings of D_{init} the Bluetooth Now switching order is always much faster than the enrollment process of FruityMesh, which is lower than the convergence delay D_{conv} since only one node needs to

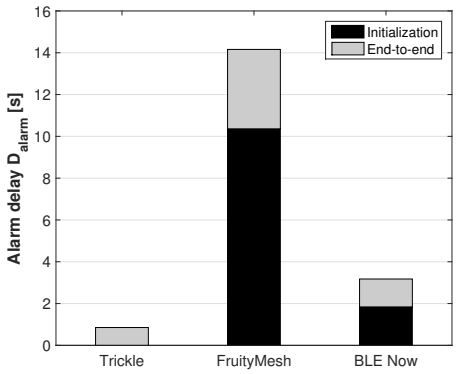

Fig. 7: Delay of the alarm when using static Trickle, static FruityMesh or Bluetooth Now. The latter offers a considerably slower initialization delay compared to FruityMesh, while using Trickle for a reduced end-to-end delay.

join the already existing network. Regarding D_{end}, Bluetooth Now performs as regular Trickle. After switching back to FruityMesh mode, the network needs to be again rebuilt (spanning for D_{conv}) before the sensor nodes can continue forwarding their traffic. This effect is not an issue if additional buffers are implemented in the sensors either to momentarily store this information until the whole network is connected or to save its state so it does not need to be rebuilt again.

The second benefit of the Bluetooth Now paradigm is a conservative power consumption scheme: when no alarm is triggered the network stays in the connected, low power state and only switches to the battery hungry flooding mode when needed. However, as the alarm delay is low this only lasts a few seconds until the network switches back again.

Note that the trade-off between delay and energy consumption will be accentuated for larger networks, as the number of hops will increase. Therefore, schemes such as Bluetooth Now are needed to ensure proper performance in forthcoming deeply populated IoT networks.

VII. Conclusion

As a primary contribution this paper presents the first measurement based multi-hop comparison between a flooding and a connection oriented mesh technology for BLE: Trickle and FruityMesh, respectively. When compared to the connection oriented scheme and under similar configuration parameters, it is obtained that the flooding scheme can trade a lower end-to-end delay for a higher energy consumption, while offering a comparable performance in terms of PDR. Based on these results, it is not clear whether BLE mesh should be based on advertising or connection events.

The official BLE mesh standard has been released prior to the measurement campaign of this work, so the Trickle characterization serves as a first proof-of-concept for BLE mesh. In turn, all its benefits and drawbacks are common to any flooding protocol. In light of these results, connection-oriented profiles may be introduced to the standard to guarantee the delivery of sensible data and to minimize the energy consumption of the network.

As a secondary contribution of this paper the Bluetooth Now paradigm is introduced. In such, the network runs in connected mode for sending periodic and non-time sensitive data to the sink, while switching to flooding for delivering sporadic and urgent data. Measurement results show that this scheme offers one order of magnitude lower delay to the urgent data while maintaining low energy consumption for the periodic data.

Autonomous mesh mode switching, as well as co-existence of flooding and connected modes running in parallel in the network are left as future work.

Acknowledgments

This work is supported by IWT CELTIC O&O project ASUA/RoCCS. Yuri Murillo is funded by a PhD fellowship of the Research Foundation Flanders (FWO).

References

[1] P. D. Marco, R. Chirikov, P. Amin, and F. Militano, "Coverage analysis of bluetooth low energy and ieee 802.11ah for office scenario," in *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2015 IEEE 26th Annual International Symposium on*, Aug 2015, pp. 2283–2287.

[2] Bluetooth SIG, "Bluetooth core specification 5," December 2016.

[3] M. E. M. Campista, P. M. Esposito, I. M. Moraes, D. G. Passos, C. V. N. De Albuquerque, D. C. M. Saade, and M. G. Rubinstein, "Routing metrics and protocols for wireless mesh networks," *IEEE Network*, vol. 22, no. 1, pp. 6–12, Jan 2008.

[4] T. Zahn, G. O'Shea, and A. Rowstron, "An empirical study of flooding in mesh networks," Tech. Rep., April 2009. [Online]. Available: https://www.microsoft.com/en-us/research/publication/an-empirical-study-of-flooding-in-mesh-networks/

[5] M-Way Solutions, "Fruitymesh project page," https://github.com/mwaylabs/fruitymesh/wiki, accessed: 2017-07-28.

[6] S. Sirur and P. J. et al., "A mesh network for mobile devices using bluetooth low energy," in *SENSORS, 2015 IEEE*, Nov 2015, pp. 1–4.

[7] S. Raza, P. Misra, Z. He, and T. Voigt, "Building the internet of things with bluetooth smart," *Ad Hoc Networks*, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870516302050

[8] H. S. Kim, J. Lee, and J. W. Jang, "Blemesh: A wireless mesh network protocol for bluetooth low energy devices," in *2015 3rd International Conference on Future Internet of Things and Cloud*, Aug 2015, pp. 558–563.

[9] P. Zenker, S. Krug, M. Binhack, and J. Seitz, "Evaluation of ble mesh capabilities: A case study based on csrmesh," in *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, July 2016, pp. 790–795.

[10] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, ser. NSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251175.1251177

[11] Nordic Semiconductor, "Nordic nrf52 development board," https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF52-Preview-DK, accessed: 2017-07-28.

[12] ——, "Nordic nrf52 online power profiler," https://devzone.nordicsemi.com/power/, accessed: 2017-07-28.